



# Specification of multiparty audio and video interaction based on the Reference Model of Open Distributed Processing

Valérie Gay <sup>a,\*</sup>, Peter Leydekkers <sup>b</sup>, Robert Huis in 't Veld <sup>c</sup>

<sup>a</sup> *Université Paris VI, Laboratoire MASI, 4, place Jussieu, 75252 Paris Cedex 05, France*

<sup>b</sup> *KPN Research, P.O. Box 15000, 9700 CD Groningen, The Netherlands*

<sup>c</sup> *Twente University (INF / TIOS), P.O. Box 217, 7500 AE Enschede, The Netherlands*

---

## Abstract

The Reference Model of Open Distributed Processing (RM-ODP) is an emerging ISO/ITU-T standard. It provides a framework of abstractions based on viewpoints, and it defines five viewpoint languages to model open distributed systems. This paper uses the viewpoint languages to specify multiparty audio/video exchange in distributed systems. To the designers of distributed systems, it shows how the concepts and rules of RM-ODP can be applied.

The ODP “binding object” is an important concept to model continuous data flows in distributed systems. We take this concept as a basis for multiparty audio and video flow exchanges, and we provide five ODP viewpoint specifications, each emphasising a particular concern. To ensure overall correctness, special attention is paid to the mapping between the ODP viewpoint specifications.

*Keywords:* RM-ODP; Multimedia; TINA-C

---

## 1. Introduction

Trends in research and commercial markets indicate a growing interest in multimedia applications that are distributed over different networks. Due to the increasing interconnectivity of these networks and the decreasing price of hardware and software, multimedia applications will soon reach the office and home environment. For these applications, audio/video exchange will be necessary between users that are geographically distributed. Moreover, as these applications will have to operate in a multi-vendor environment, they

are expected to support features as interoperability and the fast introduction of new or upgraded applications. To meet these goals, the TINA Consortium (TINA-C) was founded.

TINA-C (Telecommunications Information Networking Architecture Consortium) is a world-wide initiative of telecommunication network operators, telecommunication service providers, and vendors of telecommunication hardware and software. Their goal is to create an architecture (called TINA) that provides a framework for the specification of telecommunication based applications. TINA has to be applicable to a broad variety of telecommunication networks, such as broadband, narrowband, multimedia, and computer communication networks. TINA-C has

---

\* Corresponding author. E-mail: Valerie.Gay@masi.ibp.fr.

taken the reference model of open distributed processing (RM-ODP) as one of the starting points for the development of TINA.

RM-ODP [7–10] is an ISO/ITU-T standard-in-progress. It provides an object oriented framework for the design of open distributed systems. RM-ODP identifies five different viewpoints to handle the complexity of a distributed system. Each viewpoint represents a different view of the same system. RM-ODP defines a language for each viewpoint, consisting of concepts and rules, to model a system from that view. The languages are applicable to a broad range of distributed systems. In order to use them for a specific problem domain, they need to be specialised. Audio/video exchange between users is an important feature in the design of distributed multimedia applications. Both TINA and RM-ODP need to support it.

The goal of this paper is to show how RM-ODP can be used to specify audio/video exchange between users in a distributed environment. The specification is implemented as part of a video-phone application built at KPN Research. A specification of multiparty audio/video exchange is provided that is based on the ODP notion of stream binding.

In the area of multimedia handling and exchange, most research work addresses only the computational and engineering viewpoints (e.g. [3,4]). The contribution of this paper is the definition of a particular service using all five ODP viewpoints. The aim is to evaluate the suitability of the RM-ODP concepts and rules as described by the five viewpoint languages. For each viewpoint language, the relevant concepts and rules are applied to specify audio/video exchange. The paper confirms the suitability of the RM-ODP architecture that can be specialised by the TINA-C group.

RM-ODP is followed as closely as possible. Where necessary, additional specification techniques have been applied to support the audio/video specification (e.g. Object Modeling Technique (OMT) and an extension of Object Management Group (OMG) interface definition language called TINA Object Definition Language (TINA-ODL). ODP is not over prescriptive

which implies that several RM-ODP concepts can be interpreted differently. OMT and TINA-ODL were found suitable to describe the concepts of the information and computational languages in a more prescriptive manner. The specialisation of RM-ODP concepts used in this paper are in line with those proposed by TINA-C. Special attention is paid to mappings between the five viewpoint specifications, to ensure consistency between the specifications.

This paper is organised as follows. Section 2 introduces the stream binding concept. Each of the following sections discusses an ODP viewpoint specification of audio/video exchange. Section 3 presents an enterprise specification focusing on the requirements and objectives of the stream binding object. Section 4 provides an information specification of the binding contract. Section 5 presents a computational specification that describes in a distribution transparent way the stream binding by means of objects that exchange video and audio. The engineering specification in section 6 presents an abstract infrastructure that enables the execution of the computational specification. The technology specification in Section 7 shows a first step in realising audio/video exchange using ANSAware [2]. Finally, Section 8 evaluates the experience obtained in applying the RM-ODP viewpoint languages to the stream binding concept.

## **2. Multiparty audio/video stream binding**

The exchange of continuous media in distributed multimedia applications is rather complex. In case of a real-time multimedia conferencing application, the participants that are separated geographically, exchange real-time video and audio. The audio-visual exchange should be as natural and flexible as possible. This implies that requirements like lip synchronisation and synchronisation of display across multiple workstations need to be taken into account while specifying the multimedia conferencing application.

To fulfil these requirements, the multimedia conferencing application has stringent network

performance and synchronisation requirements on the exchange of audio and video flows. Furthermore, the number of exchanged flows and corresponding quality can change during the lifetime of the conference. This is due to the fact that the application provides control operations to join or leave the conference, and to modify the quality of service (QoS) of flows.

To address this complex functionality, RM-ODP has introduced the notion of *stream binding object* in the computational language. RM-ODP provides only the theoretical computational concept without specific refinements. In this paper, we use the stream binding object as a basis and provide five ODP viewpoint specifications of a particular stream binding object, i.e. multiparty audio/video stream binding object. This object manages the stream interfaces which are used for the real-time multiparty audio/video interac-

tions. Also control operations can be performed on the multiparty stream binding object.

Figure 1 shows the computational representation of the multiparty audio/video stream binding object and its environment that will be described using the five viewpoint languages.

The rectangle in the middle denotes the multiparty audio/video stream binding object. Its environment (grey areas) consists of application and system parts and the supporting network infrastructure. The  $\overline{\top}$  symbols denote stream interfaces via which audio/video producers and consumers exchange audio and/or video (①). The multiparty audio/video binding object manages the interactions between the stream interfaces it encompasses. It encapsulates the mechanisms that are used for this, and it abstracts away from distribution aspects. The  $\top$  symbol on top of the rectangle, denotes the stream control interface of

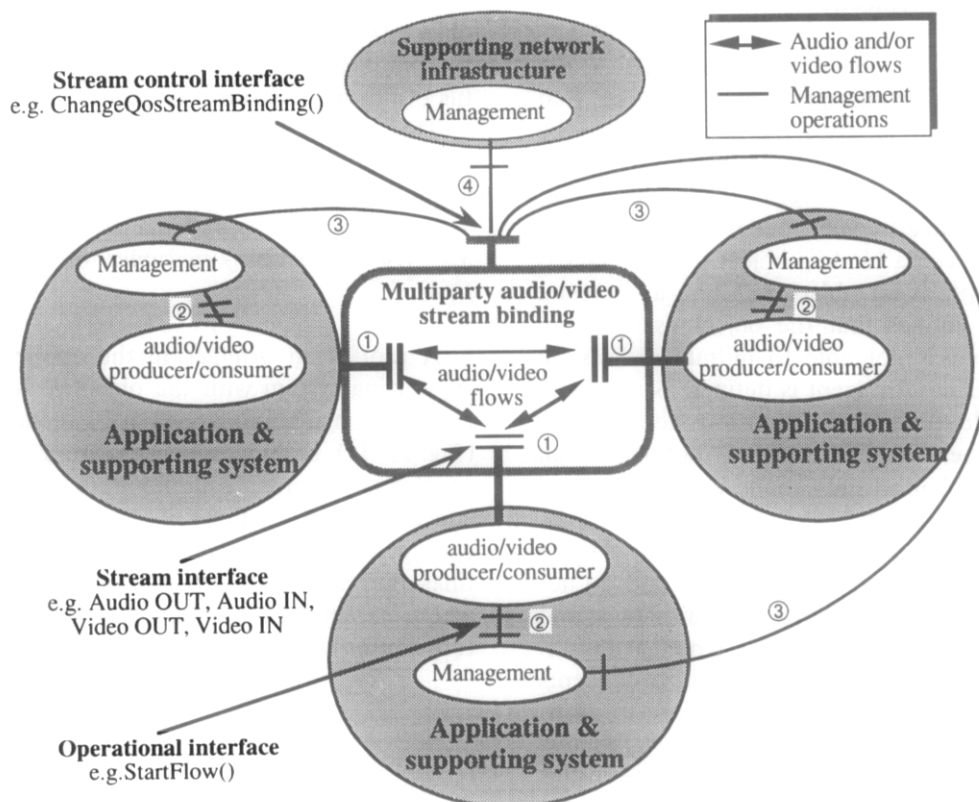


Fig. 1. Multiparty audio/video stream binding.

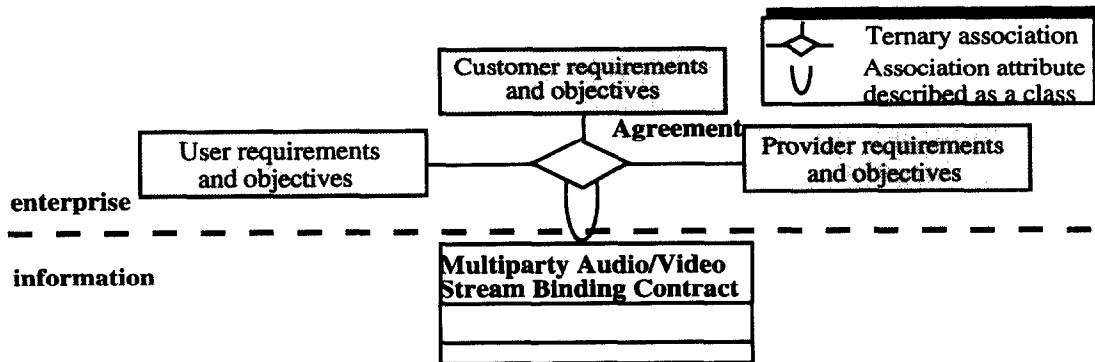


Fig. 2. The binding contract between users, customers, and provider.

the binding object. Via this interface, the multiparty audio/video stream binding object provides operations ③ ④ to the environment that controls its functioning.

### 3. Enterprise specification

The enterprise specification provides a description of the requirements and objectives that the environment imposes on the system to be designed. It justifies the design of a system. We used the ODP enterprise terms of *artefact*, *agent* and *performative actions* to describe the multiparty audio/video binding object. An artefact is defined in [9] as an object fulfilling an artefact role which implies that the object uses certain resources but is not able to initiate actions on those resources. An agent is defined as an object which is able to perform actions on the resources it uses. A performative action is defined as an action which changes obligations, prohibitions and permissions of objects.

The concepts and rules defined in the ODP enterprise language are rather vague and do not provide real support in deriving requirements and objectives of the system to be designed. We question the usefulness of these enterprise concepts because some imagination is necessary to apply them. In addition to these concepts we used a more telecommunication specific approach in this paper. We have taken the roles that stakeholders<sup>1</sup> in telecommunication can play with respect to a

service (i.e. user, customer and provider) and projected them onto the environment of the stream binding object. For the application and supporting system, this results into audio/video producer/consumer objects (users) and objects managing these users (customers). For the supporting network infrastructure, a management object (provider) is distinguished that manages the binding object in correspondence to the status of the network infrastructure.

This section models each of these objects and their requirements on the stream binding object using enterprise language concepts.

#### 3.1. User: producer/consumer of audio/video flows

A user is an *artefact* in the application and supporting system with the objective to produce and/or consume flows via its stream interface (Fig. 1, ①). Examples of users are microphones, speakers, cameras, and monitors. A user indicates the type of flows it can handle and the coding formats it requires. Furthermore, it defines the values of QoS parameters it requires. These parameters specify audio/video quality in terms of broadcast TV quality, HDTV, telephone

<sup>1</sup> Stakeholder is a telecommunication concept that denotes an organisation or person that has a commercial or regulatory interest in telecommunication services.

audio quality, HiFi or CD quality. Furthermore, QoS requirements between flows are defined that describe e.g. that lip-synchronisation between audio and video is required and that simultaneous delivery of audio/video to multiple users is desired.

### 3.2. Customer: local application and supporting system

A customer is an *agent* having the objective to manage and support the audio/video producer/consumer. It takes into account the user's policies that it has to support and manage. Customers can invoke *performative actions* (Fig. 1, ②) to create users, delete users or adjust policies of existing users.

A customer carries out *performative actions* dealing with configuration and resource management. For instance, it has to manage the priority of flows. In case of network congestion or local resource problems the flow with the lowest priority is either removed or delayed. The customer also performs end-to-end (re)negotiations with other customers to determine acceptable, preferred, and unacceptable values of QoS parameters. The latter is reflected in the "binding contract" (Fig. 2). A binding contract describes the agreed outcome of the negotiations between the user, customer, and provider which should be followed during the existence of the multiparty binding object. The customer can execute performative actions (Fig. 1, ③) on the stream binding object that are related to the management of the multiparty stream binding. These actions deal with e.g. set-up, removal, and adjustment of a stream binding in accordance with the (re)negotiated end-to-end QoS requirements. *Performative actions* can also be initiated by the stream binding object to indicate, for instance, that it cannot maintain the negotiated QoS values.

### 3.3. Provider: supporting network infrastructure

The stream binding object uses the supporting network infrastructure to exchange audio and video flows between users who are geographically

separated. The provider offers an infrastructure of connections which will be used by the stream binding object. A provider is an *agent* that manages the stream binding object in correspondence to the status of the supporting network infrastructure.

The provider is responsible for the flow topology, billing aspects, security aspects, fault management, and QoS provided by the underlying network and resources. The provider is concerned with the management of the network so that the binding contract is guaranteed. For instance, it selects an appropriate routing for the audio/video channel and reserves resources on each node on that route. If the provider can no longer guarantee the binding contract, it performs actions (Fig. 1, ④) that affect the stream binding object with respect to the (re)negotiation of the binding contract.

## 4. Information specification

The information specification of the multiparty audio/video stream binding describes the information relevant to the stakeholders in the binding. It takes into account the requirements and objectives outlined in the enterprise specification.

ODP is not prescriptive with respect to information modelling. It only distinguishes three types of descriptions: *static schema*, *dynamic schema*, and *invariant schema*. A static schema specifies relevant information at a certain point in time. A dynamic schema specifies how a static schema may evolve in time. An invariant schema specifies the properties that static schemata have in common. To describe these schemata, ODP only provides *information templates*. Therefore, we extend the information viewpoint language with the concepts and notation of OMT [12].

### 4.1. From enterprise to information

The information relevant to users, customers, and provider is specified in the binding contract. This contract is the outcome of an agreement between users, customers, and provider. It satis-

fies their requirements and objectives from the enterprise specification. Figure 2 shows the relation between enterprise and information specification in terms of OMT.

#### 4.2. Invariant schema of the binding contract

The binding contract, as shown in Fig. 2, is considered as an attribute between stakeholders and described as a single class. However, at a more detailed level, the information specification is more complex. Figure 3 shows the invariant schema of the multiparty audio/video stream binding contract.

The structure common to every contract between the customers, users, and provider is specified as an invariant schema. For a binding contract, it contains information about the users in the binding (User information objects), the stream

interfaces involved in the binding (Stream interface information objects), and the operations that stakeholders can invoke. Furthermore, the contract specifies the multiparty audio/video stream binding between stream interfaces. The information in the binding is modelled by means of the Multiparty Audio/video Stream binding information object. The invariant schema for this object specifies all the operations that stakeholders can invoke.

A user may have of one or more stream interfaces which implies that each user information object consists of one or more stream interface information objects. A stream interface consists of one or more flows which results in a stream interface information object consisting of one or more (audio, video, or composite) flow information objects. A flow information object consists of attributes indicating, amongst others the direc-

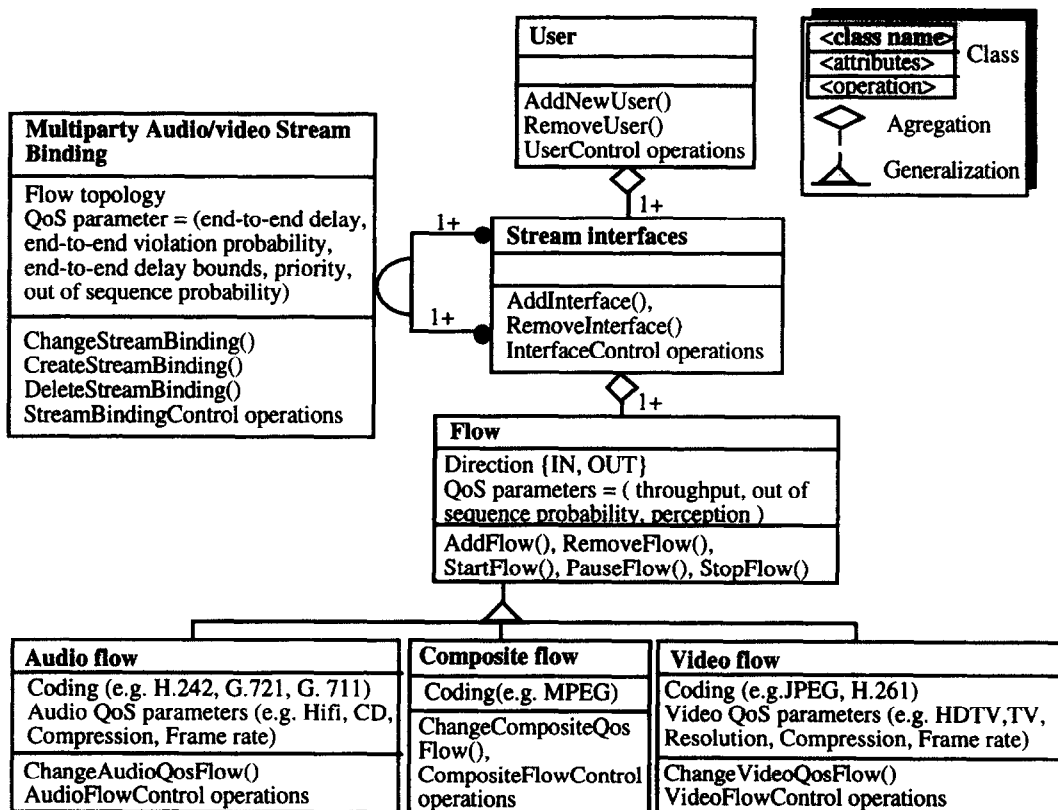


Fig. 3. Invariant schema of binding contract.

tion of the flows and QoS parameters. The QoS parameters defined in the information specification will be manipulated in the computational specification.

Information regarding the binding is captured by the multiparty audio/video stream binding information object. This information object relates two or more stream interface information objects. It contains information about the flow topology and describes the QoS that needs to be maintained while exchanging audio/video flows between interfaces.

#### 4.3. Static schema of the binding contract

A specific binding contract that exists between stakeholders at some point in time is specified as a static schema. A binding contract should satisfy its invariant schema, and consists of user information objects, stream interface information objects, flow information objects, and multiparty audio/video stream binding information objects. Other information objects can also be part of the contract, as long as they do not contradict the invariant schema.

#### 4.4. Dynamic schema of the binding contract

A dynamic schema for a binding contract has two aspects. Firstly, the effect that invoking operations have on the contract and secondly the conditions under which these operations can be invoked by the stakeholders. Both aspects are discussed below.

The *effect of operations* on the binding contract information depends heavily on the implementation choices. In general the effects can be classified into 3 categories: notification effect, negotiation effect, and no effect.

An operation having a *notification effect* is an operation by which a stakeholder informs the binding contract of a newly created computational object. The effect on the contract is that a new information object is added. For example the operation `AddNewUser` results in a new user information object, one or more stream interface information objects, and one or more flows information objects.

An operation having a *negotiation effect* is an operation by which a stakeholder negotiates with the binding object and other stakeholders about a change in the binding contract. If the negotiation is successful, changes are made to the binding contract. For example, a successful `RemoveUser` operation results in the removal of a user information object and related stream interface and flow information objects. Furthermore, the binding information object that relates the interfaces of the removed user to the interfaces of other users will be modified or removed.

An operation having *no effect* is an operation that does not affect the information objects in the binding contract, these operations are mainly control operations. For example the operation `PauseFlow` will have the effect that the concerned flow is paused. This does not have consequences for the binding contract information.

The dynamic schema also describes conditions for invoking operations on the binding contract. The invariant and static schema do not impose a specific ordering of operations that stakeholders can invoke. However, to obtain a meaningful binding contract, it is necessary to define conditions with respect to the invocation of operations. Some examples are listed below:

- A customer can only invoke an `AddNewUser` operation on an existing stream binding.
- Customers can only invoke `ChangeAudioQosFlow` and `RemoveFlow` on existing flows.
- Customers and providers can only invoke a `ChangeStreamBinding` operation or a `DeleteStreamBinding` on an existing binding.

## 5. Computational specification

The computational specification describes how distributed applications and their components are structured in a distribution transparent way. This implies that the structuring of applications is independent of computers and networks on which they run. In the computational viewpoint a distributed application consists of a collection of computational objects. A computational object provides a set of services that can be used by other objects. An object offers a computational

Table 1

TINA-ODL specification of MultiPartyStreamBindingObject

---

```

object template MultiPartyStreamBindingObject;
typedef AVuserId.... /* This type identifies uniquely the audio/video users involved */
typedef enum Direction {in, out} /* Each flow has a direction In or Out */
typedef struct { /* Definition of video flow */
    enum Coding {MPEG, JPEG, H.261,...};
    enum Resolution {640 × 480, 320 × 480,...};
    enum Color {B/W, 24 Bits, 16 Bits, RGB,...};
} videoFlow;
typedef struct { /* Definition of audio flow */ audioFlow;
typedef struct { /* Definition of combined audio and video flow */
    enum Coding MHEG;
    enum CompressionFactor {1,3,5,...};
} combinedFlow;
/* A flow is of type audio, video or combined audio/video and has a direction in or out */
typedef struct {    union FlowSort switch (FlowType) {
                    case 1: audioFlow audio;
                    case 2: videoFlow video;
                    case 3: CombinedFlow acombinedflow;
                    };
                Direction dir
            } Flow;
typedef struct {
                AVuserId                ProducingId;
                AVuserId                ConsumingId;
                Flow                    ConcernedFlow;
                integer                 NumberOfFlow;
                                        /* Unique identifier of flow */
            } FlowBindingId;
typedef sequence < FlowBindingId > StreamBindingId;
initialisation
void init (out StreamBindingControlInterface StrmCntrlInterf)
supported interfaces
                StreamBindingControlInterface = StrmCntrlInterf;
behaviour
    "An Instance of this object binds two or more stream interfaces"

```

---

Table 2

TINA-ODL specification of StreamBindingControlInterface

---

```

interface template StreamBindingControlInterface; /* ②③, operational interface type */
typedef sequence < Flow > StreamInterface;
operations
transactional void ChangeQoSStreamBinding (in StreamBindingId Binding,
                                           in QoS RequestedQoS, out QoS ProvidedQoS);
transactional void RemoveStreamBinding (in StreamBindingId Binding,
                                           out StreamBindingId RemainingBindings);
transactional void AddNewUser (in AVuserIdNewuser, in StreamInterface NewFlows,
                                in QoS RequestedQoS, out QoS ProvidedQoS, out ResultReport StatusBinding);
/* Additional stream binding control operations are possible determined by the application,
   system and network management part. */
behaviour
    "An instance of this interface template provides other objects to perform control actions on the Multiparty stream binding object."

```

---



interface to enable other objects to access a service. This is the only means by which other objects can use the service. ODP provides templates for the specification of computational objects and interfaces [9]. These templates enable the application designer to express precisely the constraints on the interfaces and objects to be designed.

Furthermore, the computational language imposes constraints on the binding of objects. Before objects can interact a binding between the objects must exist. Upon binding of computational objects, interface compatibility is checked. Furthermore, a binding must satisfy the environment contract of all the computational objects involved. An environment contract expresses constraints on the computational objects in relation to its environment and include QoS constraints, management constraints and usage constraints.

### 5.1. From information to computational

The mapping between information objects and computational objects is not necessarily a one-to-one mapping. The correspondences between the information and computational specifications must be specified in each case so that consistency between the specifications can be ascertained. The grouping of classes into objects is a decision taken by the service designer. Distribution aspects need not be taken into account at this stage.

Figure 4 indicates the mapping of several information classes on computational interfaces. For the control interface (Fig. 4, ③ ④) of the multiparty audio/video stream binding the operations of the multiparty audio/video stream binding class are taken into account. The operational interface of the audio/video producer/consumer (②) will be reflected in the operations defined in the Flow class. The stream interface (①) has the characteristics of the attributes of the Flow subclass. The attributes specified in the information specification will be reflected into parameters in computational operations. Operations and parameters naming in both viewpoints are independent.

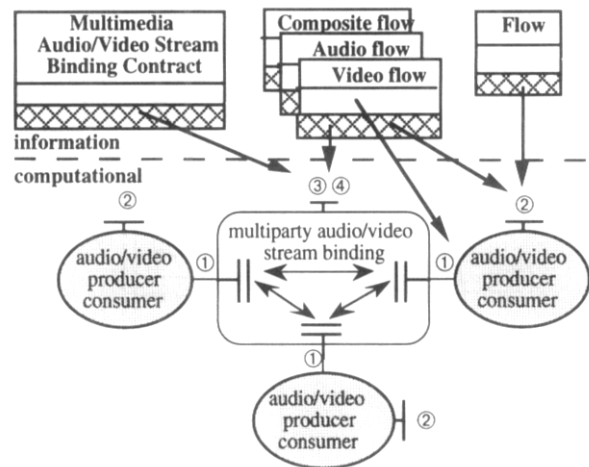


Fig. 4. Information to computational mapping.

### 5.2. Specification in TINA-ODL

RM-ODP describes a computational model applicable for distributed applications, but it does not provide a specific specification language for computational objects and interfaces. Therefore, an additional specification language, TINA-ODL, is used here to derive a computational specification of the multiparty stream binding object (Table 1) and its stream binding control interface (Table 2). TINA-ODL [6,11] provides the means to express telecommunication and multimedia oriented computational specifications. The derived TINA-ODL specifications are based on the information specification.

### 5.3. Computational choice for the multiparty audio/video exchange configuration

The multiparty audio/video stream binding can be refined to arrive at an engineering configuration. Several solutions are possible but we adapted one that is used in many distributed multimedia systems. Several implementations of distributed multiparty systems have a functional component, referred to as audio/video controller and dispatcher, that manages audio/video flows [1,5]. It receives all audio/video flows of the producers and reflects the flows (after possible manipulation) to all consumers. We adapted this

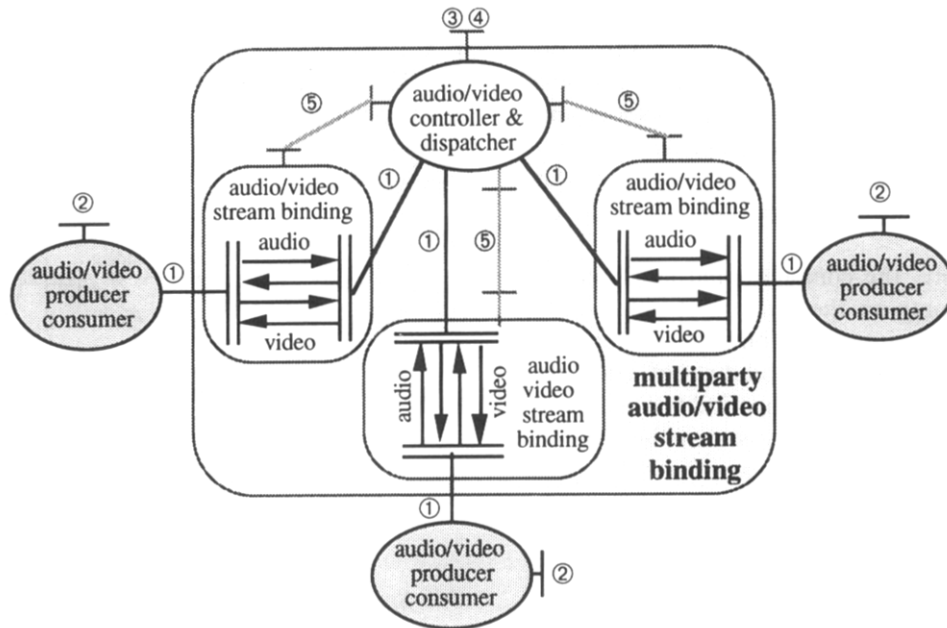


Fig. 5. Objects involved in a multiparty audio/video exchange.

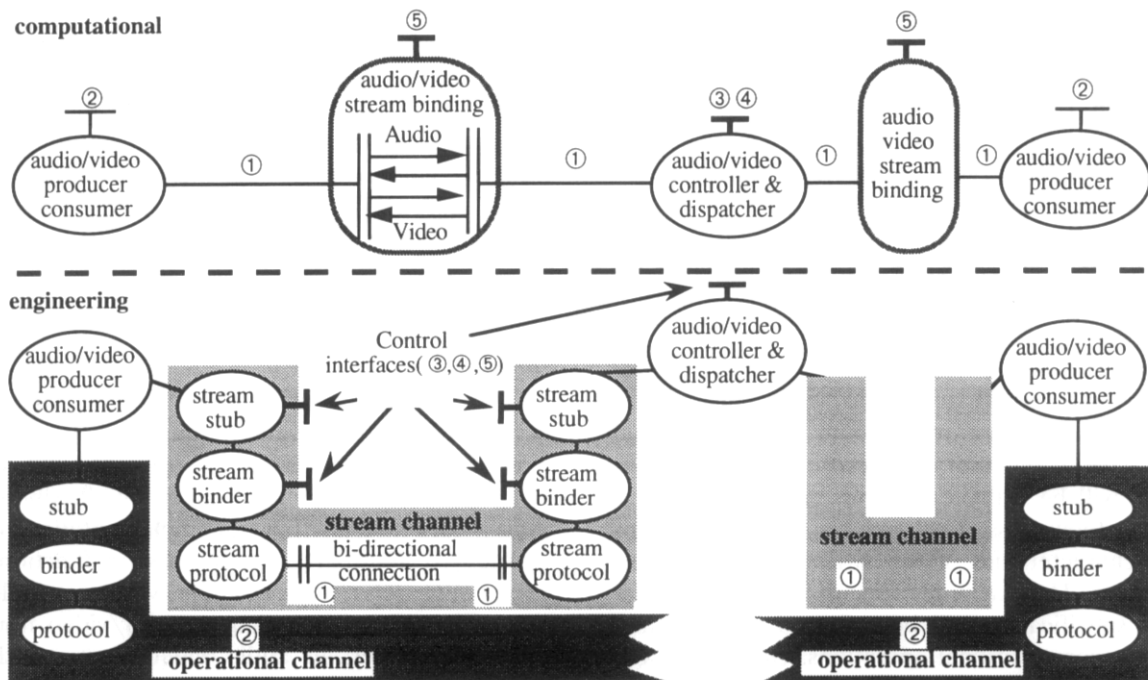


Fig. 6. Computational to engineering mapping.

approach for the multiparty audio/video stream binding as illustrated in Fig. 5.

The audio/video controller and dispatcher is in charge of redirecting stream control operations ③, ④ to each sub-stream binding ⑤. It also deals with the set-up, control and release of the audio/video bindings between the producers and consumers. It negotiates the requirements and objectives identified in the enterprise specification (e.g. encoding and compression algorithms, frame rate). An audio/video stream binding links each of the audio/video producer/consumer object to the audio/video controller and dispatcher.

## 6. Engineering specification

An engineering specification of a distributed system describes an infrastructure enabling the execution of a computational specification. The engineering language of RM-ODP describes how to structure the computational objects in order to execute them on the infrastructure. Furthermore, the functionality of objects supporting distribution transparencies are identified. This section provides an engineering specification of the multiparty audio/video stream binding object as defined in the computational specification.

### 6.1. From computational to engineering

The computational specification has to be mapped onto an engineering specification to be executed. This engineering specification preserves the behaviour described in the computational specification.

The engineering transformation of a *stream interface* (Fig. 6, ①) leads to the creation of a producer/consumer stream channel specialised for continuous flows. The QoS parameters associated with the audio/video flow defined in the computational specification influence the choice of the stream channel components.

An *operational interface* (Fig. 6, ②) is reflected in the engineering specification as a client-server channel configuration, as defined in RM-ODP. The environment constraints specific for the interfaces (e.g. security), are taken into

account while establishing a channel between the concerned objects.

The computational *stream binding control interface* (Fig. 6, ③ ④ ⑤) is located in different nodes in the engineering representation. Their communication (not reflected in Fig. 6) occurs through control channels that have the same structure as operational channels. Furthermore, supporting objects may be created (e.g. synchronisation object) to manage and control a set of inter-related stream channels.

The *audio/video producer and consumer object* as well as the *audio/video controller and dispatcher object* are transformed into basic engineering objects. If those objects are distributed over different nodes, further decomposition has to be made and several objects and channel parts must be created. Figure 6 illustrates the mapping of two computational audio/video producers and consumers onto an engineering specification.

### 6.2. The engineering stream channel

The ODP channel concept provides the engineering mechanisms to assure distribution transparent interactions between basic engineering objects. The channel consists of three engineering objects that are protocol, binder, and stub objects. Two different types of information are transported through the channels. First, the control operations enabling for example QoS negotiation. These operations require small bandwidth but they demand high reliability. Second, the real-time interactions such as voice and video exchange that need high bandwidth but a lower reliability [1]. Therefore, the channels are divided into an operational channel and stream channel, each with their own characteristics. This section presents first the concepts of stub, binder and protocol objects and describes then how a particular stream channel is established

The **stub object** provides adaptation functions to support interaction between basic engineering object interfaces in different nodes. For operational invocations the stub object provides marshalling/unmarshalling of operation parameters to enable access transparent interactions. Streams require different functionality of the stub object

due to the different nature of information that is exchanged. It should provide the mechanisms to encode and decode video/audio information. Furthermore, data available for the audio/video producer or consumer should be notified and the stream stub objects provide controlling operations to local resources (e.g. increase buffer-size) and notification of events concerning the stream (e.g. QoS changes, no buffer space available, data drop out, etc.). A stub object has a presentation interface for use by the object that is bound to the channel and a control interface for e.g. QoS management.

**Binder objects** interact with one another to maintain the integrity of the binding. Information is maintained about the channel. Binder objects are also responsible for validating the interface reference and for interacting with the relocater object to recover information about the interface location after a binding error. For streams, information is maintained with respect to the required QoS. A binder has a control interface which enables changes in the configuration of the channel and destruction of all, or part, of the channel.

The **protocol object** assures that computational objects can interact remotely with each other. Protocol objects are needed if the computational objects that have to be bound are located in different nodes. In general, the RPC mechanism is used where an operation is sent in the form of a message to a remote protocol object that is able to receive it. The object for which the call is meant executes the procedure and sends back a reply message. For the multiparty video/audio stream binding the RPC type of protocol objects are suitable for computational objects that invoke operational invocations e.g. for the control/management channel (Fig. 6, ②③④⑤). However, for the exchange of continuous flows, another specialised protocol (called stream protocol) without the RPC mechanism is necessary. RPC requires that each buffer of data to be transferred is treated as a separate action with no particular relationship between previous and future RPC calls. Continuous flows require relationships between calls and a stream protocol is applied which creates a virtual channel between two protocol objects for the duration of

audio/video flow exchange. In this case, relations between audio/video data can be defined specifically. Protocol objects can interact with objects outside a channel to obtain the information they need (e.g. Trader).

The **establishment of a stream channel**, as shown in Fig. 6, between a audio/video producer/consumer and the audio/video controller and dispatcher is made in several steps. The audio/video producer/consumer object initiates the configuration of a channel by interaction with its nucleus. The interaction syntax is.

**InitChannel** (*StreamChannel*, *producer / consumer*, *IFAVPC*, *result IFrefStreamchannel*)

where *StreamChannel* is the type of channel to create. *Producer / consumer* indicates that the audio/video producer/consumer will be both the producer and consumer role for the stream channel. *IFAVPC* is the interface of the audio/video producer/consumer object to be bound to the stream channel.

When this interaction occurs, the nucleus creates a stub object, a binder object and a protocol object corresponding to the channel type and the role. These objects are bound to create a first part of a stream channel. The stub object presentation interface is bound to the audio/video producer/consumer interface. The stub object is then bound to the binder object that is bound to the protocol object. The result of this interaction is an interface reference (*IFrefStreamchannel*). The interface reference *IFrefStreamchannel* will be communicated to the objects that want to bind to the channel. In our case the interface reference of the channel is communicated to the audio/video controller and dispatcher. The audio/video controller and dispatcher interacts with its nucleus to bind to the channel by means of the following interaction.

**BindChannel** (*StreamChannel*, *1*, *producer / consumer*, *IFAVCD*, *IFrefStreamchannel*)

where *StreamChannel* is the type of channel. *Producer / consumer* indicates that the audio/video controller and dispatcher will perform both the producer and consumer role for

the stream channel, *IFAVCD* is the audio/video controller and dispatcher object interface to be bound to the stream channel.

The nucleus determines from the interface reference *IFrefStreamchannel* the channel type and location of the protocol objects for the other participants in the stream channel. The nucleus creates a stub object, a binder object and a protocol object corresponding to the channel type of the other participants and the role. These objects are bound to the stream channel and the audio/video producer/consumer object. Then the binder objects in the channel interact with each other to enable communication across the channel. Other participants may bind to the existing channel using the same **Bindchannel** () interaction.

## 7. Technology specification

The technology language expresses how the specifications for an ODP system are implemented. This viewpoint is of interest to those responsible for the hardware and software of a distributed system, as well as, the configuration, installation, and maintenance. This viewpoint deals with local operating systems, input-output devices, storage, communication protocols etc. Specific hardware and software choices are made in this viewpoint influenced by the other viewpoint specifications. This section describes an ini-

tial implementation of a stream binding using ANSAware which is an implementation of the ANSA Architecture. The ANSA architecture is closely linked to the ODP standardisation work. The technology specification presented here corresponds to an implementation of a multiparty video phone service which enables end-users to exchange audio-visual information via their desktop computer [13]. Users can be added or removed dynamically from the ongoing session.

### 7.1. From engineering to technology

The technology specification is determined by the other viewpoint specifications but the implementation is mainly based on the engineering specification. The technology specification consists of a description of hardware and software that can implement the engineering specification taking additional enterprise requirements into account (e.g. which hardware and software is available).

Figure 7 indicates a mapping between engineering objects and technological solutions. It focuses on the implementation of the stream channel and operational channel and shows how the engineering objects are realised in hardware and software components. ANSAware is used as a distributed platform that provides the advantage that several engineering mechanisms are realised (e.g. operational interfaces are supported by ANSAware).

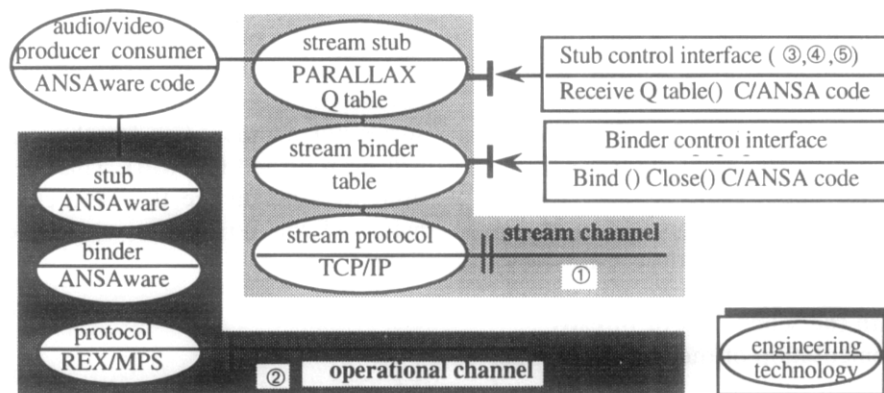


Fig. 7. Engineering to technology mapping

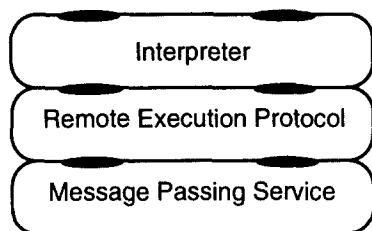


Fig. 8. Refinement of the operational protocol object based on ANSAware

### 7.2. Technology implementation of operational and stream channel

The implementation of the **operational channel** is fully supported by ANSAware. The *stub* and *binder* object are generated automatically using the ANSAware tools. The *protocol* object consists of the communication stack as shown in Fig. 8. The communication facilities of ANSAware are provided at three layers of abstraction. The highest level is the interface of the interpreter module followed by the interface of the execution protocol and at the lowest level the message passing service (MPS). The MPS service just sends messages contained in a buffer to an endpoint. Currently, message passing services are implemented on top of Interprocess Communication (IPC), Unreliable Datagram Protocol (UDP) and Transmission Control Protocol (TCP). All message passing services have the same interface and differ in the service provided. E.g. the MPS based on IPC can only be used for communication between capsules residing on the same node. The execution protocol adds extra functionality to the MPS. This includes fragmenting of messages and retransmission in case of message loss. Furthermore, this protocol ensures that invocations are delivered at the right interface. Currently, ANSAware supports the Remote Execution Protocol (REX). This protocol is an implementation of the Remote Procedure Call (RPC) mechanism. The interpreter integrates the execution protocol with the scheduling of threads.

For the implementation of the **stream channel** ANSAware has been extended with a number of functions to support continuous data exchange.

Based on the available hardware<sup>2</sup> and software to exchange compressed video, a specific implementation is given to the stream stub and stream binder object.

The stream stub object maintains a Parallax quantisation table (Q-table). A Q-table contains data for the Parallax video cards with respect to the compression of video images which is needed to display video images. A Q-table is created by a producer of video images. The control interface of the stream stub object is used to exchange Q-tables.

The stream binder object is responsible for the mapping of dataflow identifiers onto an endpoint identifier (sockets) that is created by the stream protocol object.

The stream protocol object is based on TCP/IP. It transports continuous dataflows between nodes. Currently, the ANSAware protocol supports only remote operations by means of a RPC mechanism. For the stream protocol, the RPC mechanism is excluded and operations invoked on the stream protocol object interface are directly passed to the TCP protocol. The stream protocol provides control operations to create, connect and close endpoints. If two endpoints are connected, the stream protocol ensures that data written at one endpoint is delivered to the other endpoint. The operation `Create_Endpoint` operation will initiate a TCP socket to listen for a connection request. If a connection is accepted, a function will be attached to the socket and incoming data will be read by that function and delivered to the audio/video producer/consumer object via the stream binder object and stream stub object. The `Connect_flow` operation attempts to make a connection to a socket instantiated by a `Create_Endpoint` operation. If this attempt is successful an identifier for the established flow will be returned (`FlowId`). This identifier can be used to write on the flow and close the flow by means of the `Write_flow` and `Close_flow` operations.

<sup>2</sup> Sparc workstations extended with Parallax video cards for the compression and displaying of video images.

The stream protocol is implemented by means of the TCP/IP protocol due to the available technology for this implementation. Other transport protocols, more specific for real-time continuous dataflows, will be used in future versions of the multiparty video phone application.

## 8. Conclusions

This paper has shown how the RM-ODP standard can be used to specify multiparty audio/video exchange in a distributed environment. The multiparty audio/video stream binding is outlined using computational terms and then five corresponding viewpoint specifications are provided. This approach is not part of RM-ODP. Although this approach worked out fine for the multiparty audio/video stream binding, further research is needed to evaluate its suitability for other types of problem domains.

We experienced difficulties with the enterprise specification. This language is too vague to evaluate its suitability. This paper gave an enterprise specification by enriching it with additional concepts like the roles of *stakeholders* (user, customer, provider). Nevertheless, this specification remains rather artificial. Guidelines are needed that assist the service designer to derive requirements and policies for the system to be designed.

The information and technology viewpoint languages are not prescriptive enough but suitable if the application designer chooses the proper specification methods (e.g. OMT for the information viewpoint) or the proper technology (e.g. multimedia protocols). For the information specification, a number of relations were introduced between classes of the invariant schema. These relations are derived from a set of basic relations of OMT. These relations are usually parameterised with text to make their meaning precise. Whether they are sufficiently expressive for the specification of other types of distributed systems, remains for further research.

The computational and engineering languages were found suitable for the specification of multimedia systems. For the computational language we just had to choose a particular language

(TINA-ODL) to describe computational objects and interfaces more precisely. In the computational specification, we introduced the audio/video controller and dispatcher to handle the binding. Distribution aspects are not taken into account at the computational level but it is possible to implement the controller and dispatcher in a distributed way at the engineering level. However, we chose not to introduce this additional complexity in this paper. For the engineering specification, it was necessary to introduce a specialised stream channel to transport continuous flows. A configuration of objects is presented for the support of the multiparty audio/video stream binding, including objects to control and coordinate multiple stream channels.

The implementation described in the technology viewpoint was used to validate the modelling process along the ODP viewpoints. It is likely that more specific hardware/software will be used in a operating environment to meet e.g. the strict performance requirements on multimedia applications. Further research is needed to evaluate the suitability of other existing protocols that can be found in e.g. OSI or ATM. Also attention has to be paid to the implementation of the stream stub and stream binder object.

Our main goal was to show the suitability of the viewpoint languages for multimedia system design. RM-ODP states that the five viewpoints are not independent and each viewpoint presents a partial view of the complete system specification. A set of consistency constraints between pairs of terms in two viewpoint languages are defined in [9]. Consistency checking is an important part of demonstrating the correctness of the complete set of specifications. But in the current version of the RM-ODP standard, the links between the viewpoints are not clear at all. This paper presented our view how the viewpoints can be related during the specification process of the multiparty audio/video stream binding.

## References

- [1] L. Aguilar, J.J. Garcia-Luna-Aceves, D. Moran, E.J. Graighill and R. Brungardt, *Architecture for a Multimedia Teleconferencing System*. ACM, 1986.

- [2] APM Ltd., *The ANSA Programmers Manual*, Cambridge, England, 1991.
- [3] G.S. Blair, G.Coulson, P. Auzimour, L. Hazard, F. Horn, and J.B. Stefani, An integrated platform and computational model for Open Distributed Multimedia applications, *3rd Int. Workshop on Network and Operating System Support for Digital Audio and Video*, November 1992.
- [4] G. Coulson, G.S. Blair, N. Davies and N. Williams, Extension to ANSA for multimedia computing, *Comput. Networks ISDN Systems* **25** (1992) 305-323.
- [5] T. Hiroya, A. Tomohiko, M. Shigeki and S. Kazunori, Personal multimedia-multipoint teleconferencing system, *Proc. IEEE INFOCOM'91*, 1991.
- [6] N. Natarajan et al., Computational Modelling Concepts; TINA-C deliverable, TP-A-2.-NAT-.002\_5.0\_93 edition, October 1993, restricted distribution.
- [7] Basic Reference Model of Open Distributed Processing, part 1: overview and guide to use, ITU/T X.901-ISO N7053-1, November 1993.
- [8] Basic Reference Model of Open Distributed Processing, part 2: descriptive model (DIS), ITU/T X.902-ISO 107046-2, February 1994.
- [9] Basic Reference Model of Open Distributed Processing, part 3: prescriptive model (DIS), ITU/T X.903-ISO 107046-3, February 1994.
- [10] Basic Reference Model of Open Distributed Processing, part 4: architectural semantics, ITU/TS X.904-ISO N7056, June 1993.
- [11] The Common Object Request Broker: Architecture and Specification, OMG Document Number 91.12.1, Draft edition, December 1991.
- [12] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *Object-Oriented Modelling and Design*, Prentice-Hall, Englewood Cliffs, N.J. (1991).
- [13] A.T. van Halteren, Realization of a stream object in ANSAware, University of Twente, Department of Computer Science (TIOS), June 1994.



**Valérie Gay** is associate professor and she joined the Networks, Architectures and Distributed Applications department of the French MASI laboratory in January 1987. She received her Ph.D. degree in computer systems from the University of Paris 6 in May 1991. Currently, her research activities include distributed application architectures, multimedia, and Open Distributed Processing (ODP). She participates to the French AFNOR

standardisation group on RM-ODP since 1988 and is an ISO delegate for the SC21/WG7 on RM-ODP since May 1991.



**Peter Leydekkers** joined the Dutch KPN Research in 1990. He received his M.S. degree in computer science from the University of Twente in 1989. He is preparing a Ph.D. thesis in the area of Open Distributed Systems and is currently working for one year in the TINA Core Team (USA). His research activities include Open Distributed Processing (ODP), TINA architecture and the design of distributed multimedia services in an

open telecommunication environment. He participates to the Dutch NNI standardisation group on RM-ODP and is ITU/ISO delegate for SC21/WG 7 on RM-ODP since 1991.



**Robert Huis in 't Veld** is a research associate at the Dutch TeleInformatics and Open System group of the University of Twente. He joined this group in April 1992. He received his M.S. degree in computer science from the Eindhoven University of Technology in 1987. He received his Ph.D. degree at the Eindhoven University of Technology in December 1994. His research activities include Open Distributed Processing (ODP), Integrated Service Engineering, application of formal languages, and the design of multimedia services and systems. He participates to the Dutch NNI standardisation group on RM-ODP since 1992.

standardisation group on RM-ODP since 1988 and is an ISO delegate for the SC21/WG7 on RM-ODP since May 1991.